Tampereen yliopisto

Ville-Pekka Frantsi

TAVALLISIMMAT WEB-OHJELMOINNIN TIETOTURVAONGELMAT

Katsaus JavaScript-kielen ja AngularJS-työkalun yleisimpiin turvallisuusongelmiin

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Toukokuu 2019

TIIVISTELMÄ

Ville-Pekka Frantsi: Tavallisimmat web-ohjelmoinnin tietoturvaongelmat
Kandidaatin tutkinto
Tampereen yliopisto
Tekniikan kandidaatti
Toukokuu 2019

Tässä kandidaatin tutkielmassa esitellään web-ohjelmoinnin yleisimpiä ongelmia JavaScript-ohjelmointikielen näkökulmasta ja tutkitaan miten AngularJS-työkalun käyttö kehityksessä vaikuttaa ongelmien välttämiseen tai ratkaisemiseen.

Tutkielmassa tutustutaan JavaScript-kieleen ja tietoturvaan yleisesti ja käydään läpi tarkemmin muutamia tärkeitä heikkouksia JavaScript-kielen tietoturvallisuudessa. Tutkielmassa esitellään AngularJS-työkalu ja kuinka sen käyttö voi helpottaa verkkokehitystä.

AngularJS-työkalu sisältää monia sisäänrakennettuja suojia JavaScript-kielen tyypillisimmille ongelmille, kuten XSS-hyökkäyksille. Nämä suojat eivät kuitenkaan ole täydellisiä, sillä ohjelmointivirheet sekä huolimattomuus voivat tehdä jopa AngularJS-työkalulla luodusta ohjelmasta haavoittuvaisen. Työkalujen käyttö on kuitenkin suositeltavaa verkkokehityksessä, sillä oikein käytettynä työkalut tekevät verkkokehityksestä nopeampaa, sulavampaa ja turvallisempaa.

Avainsanat: JavaScript, tietoturva, XSS.

ABSTRACT

Ville-Pekka Frantsi: The most common information security problems in web development
Bachelor's thesis
Tampere University
Bachelor of Science in Technology, B.Sc.
May 2019

This bachelor's thesis introduces the most common problems of web programming from the point of view of the JavaScript programming language and examines how the use of AngularJS tool in development influences avoiding or solving the problems.

The thesis introduces the JavaScript language and information security in general and goes through some of the most important weaknesses in the information security of the JavaScript language. The thesis introduces the AngularJS tool and how its use can facilitate network development.

The AngularJS tool includes many built-in shields for the most common problems with JavaScript, such as a way to shield against XSS attacks. However, these protections are not perfect, and as such programming errors and negligence can make even a program created with the AngularJS tool vulnerable. However, the use of tools is advisable in web development, as properly used tools make network development faster, smoother and safer.

Keywords: JavaScript, information security, XSS.

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

SISÄLLYSLUETTELO

.

KUVALUETTELO

1. JOHDANTO

Verkkoteknologioita on hyödynnetty jo kymmeniä vuosia niiden kehityksen kasvun tahdin kiihtyessä jatkuvasti. JavaScript ja sen lukuisat työkalut sekä niiden käyttäjät ovat sekä kärsineet kehityksen nopeudesta että hyötyneet kehityksen tarjoamista mahdollisuuksista. Uudet mahdollisuudet tuovat kuitenkin mukanaan ongelmia, mikäli palvelujen kehittäjät eivät tutustu tarpeeksi hyvin työkalujen uusiin ominaisuuksiin ja erityisesti niiden turvallisuuteen ja sen aukkoihin.

Nykypäivänä uusimpia verkkoteknologioita hyödyntävät sivustot turvautuvat yhä enemmän ja enemmän erilaisiin skripteihin, eli suoritettaviin koodinpätkiin muuntaakseen yksinkertaisista sivuista monimutkaisia palveluita. Usein palveluntarjoajat käyttävät sivuillaan kolmannen osapuolen tuottamaa koodia ostettuna tai lainattuna palveluna. [1] Erilaisia palveluita, jotka koostuvat itse monista erilaisista pienemmistä paketeista on tarjolla runsaasti ja helposti, joskus jopa ilmaiseksi. Palveluiden hankkimisen helppous ja niiden runsaus aiheuttavat ongelmia JavaScriptin kanssa sen dynaamisten ominaisuuksien kautta: JavaScriptiä ei tarvitse kääntää, vaan sitä voi suorittaa suoraan esimerkiksi käyttäjän syötteestä. [2]

Tässä kandidaatintyössä on tarkasteltu tavallisimpia verkko-ohjelmointiin liittyviä tietoturvavirheitä JavaScriptiä käytettäessä. Toisessa luvussa esitellään yleisesti tietoturvan kehitys ja miksi tietoturva on tärkeää. Kolmannessa luvussa esitellään JavaScript yleisesti. Neljännessä luvussa tutustutaan JavaScriptin yleisimpiin tietoturvaongelmiin ja pohditaan, voitaisiinko työkaluilla helpottaa näiden ongelmien ratkaisemista. Viidennessä luvussa esitellään yleisesti AngularJS, yksi JavaScriptin suosituimmista työkaluista sekä pohditaan syitä sen suosioon tietoturvan näkökulmasta. Kuudes luku käsittelee työkalujen ja tietoturvaongelmien suhdetta toisiinsa; kuinka suuri osa tietoturvaongelmista voidaan laittaa kielessä tehtyjen suunnitteluratkaisujen tai kielen ominaisuuksien syyksi ja kuinka suuri osa johtuu käyttäjän tai työkalun ongelmista. Viimeinen luku kokoaa yhteen tärkeimmät havainnot ja johtopäätökset.

2. TIETOTURVA

Nykypäivänä eräs isoimmista ongelmista tietoturvan suhteen ei ole enää teknologian riittämättömyys, vaan sen oikeat käyttökohteet ja oikeaoppinen käyttöönotto. Liian usein käytetään sopimatonta tai riittämätöntä teknologista ratkaisumallia tietoturvaongelman ratkaisuun. Yleisenä ongelmana tietoturvan saralla on myös tietämättömyys mahdollisista uhista ja niiden torjumisesta; kuinka torjua ongelma, jonka olemassaoloa ei edes tiedosteta? Tämän takia on tärkeää, että tietoturvan perusteet ovat hallussa myös henkilöillä, jotka eivät ole varsinaisessa kehitystyössä mukana.

Tieto itsessään on jo monimuotoinen ja monitulkintainen termi; esimerkiksi tietojohtamisessa tieto jaetaan kolmeen eri osaan: tietämys, informaatio ja data. Näiden kolmen osan lisäksi tieto jaetaan hiljaiseen ja eksplisiittiseen tietoon. Hiljainen tieto käsittää henkilön kokemuksen ja oppiman kautta saadun tietämyksen. Eksplisiittinen tieto on talletettua, kirjoitettua hiljaista tietoa, esimerkiksi kielet. [22]

Tiedon lailla myös tietoturva käsitteenä on varsin laaja, mutta siinä on muutamia yleisesti tunnistettavia käsitteitä ja periaatteita. Näistä yksi on CIA-malli, eli *Confidentiality, Integrity, Accessibility*, tai suomeksi luottamuksellisuus, eheys ja saatavuus. Jotkin tahot lisäävät näiden kolmen tekijän lisäksi vielä muutamia lisäyksiä, esimerkiksi ISO-standardin, Yhdysvaltojen lain ja monien artikkelien välillä on eroja termien määrittelyssä [21].

Luottamuksellisuudella tarkoitetaan tarvetta suojata informaatiota tai dataa ulkopuolisilta tai riittämättömän pääsyn omaavilta tekijöiltä. Tiedon luottamuksellisuus voidaan taata esimerkiksi käyttämällä järkevästi konfiguroitua pääsynvalvontaa ja varmistamalla, etteivät ulkopuoliset pääse käsiksi tietoon esimerkiksi sosiaalisen manipuloinnin kautta [20]. Muita tapoja varmistaa tiedon luottamuksellisuus ovat tiedon salaus käyttäen salausalgoritmeja tai antamalla käyttöön vain osa tiedosta esimerkiksi poistamalla salatun tiedon jaettavasta dokumentista [20].

Eheydellä tarkoitetaan tarvetta pitää tieto muuttumattomana ja oikeellisena eri tahojen välillä sekä pitää tiedon alkuperä saatavilla ja muuttumattomana [20]. Tiedon muuttumattomuudella tarkoitetaan sitä, että tieto ei muutu missään vaiheessa: siihen ei lisätä mi-

tään, sitä ei muokata tai siitä ei poisteta mitään. Tiedon eheyttä voidaan suojella seuraavilla tavoilla: pääsynvalvonta ja muokkausoikeuksien rajoittaminen sekä tiedon muokkauksen valvontamekanismit [20].

Saatavuudella pyritään varmistamaan, että tieto on käytettävissä tietyin, erikseen määritetyin kriteerein, esimerkiksi tietyn ajan sisällä. Saatavuuden kriteerien tiukkuus riippuu merkittävästi tiedon laadusta ja tietoa tarvitsevan tahon tärkeydestä [20]: esimerkiksi verkkosivujen tapauksessa latausaikojen tulee olla mahdollisimman nopeita, mutta yliopiston opintosuoritusten kirjauksessa ei ole kiirettä. Tiedon saatavuus voidaan jakaa yleisesti kolmeen alalajiin: rautapuolen saatavuus, ohjelmistollinen saatavuus sekä tiedon saatavuus [20]. Kaikkia kolmea saatavuuden alalajia ei voida luonnollisesti taata samoilla tekniikoilla, vaan jokaisen alalajin saatavuuden takaamiseen löytyvät omat tapansa.

Englanniksi termi *information security* on huomattavasti laajempi verrattuna suomen kielen termiin tietoturva. Englanninkielisellä termillä voidaan tarkoittaa suomen kielen tietoturvan merkitystä tai muitakin termin kattavia asioita, joten on tärkeää rajata termiä ja ilmoittaa kontekstin puitteissa, millaisesta tietoturvasta puhutaan.

3. JAVASCRIPT YLEISESTI

JavaScript on vuonna 1995 kehitetty ohjelmointikieli, jonka ensimmäinen versio kehitettiin Netscape Navigator 2 -selaimelle. Verkkoselaimilla ei ollut minkäänlaisia mahdollisuuksia suorittaa nykypäivän monimutkaisia skriptejä ennen JavaScriptin kehittämistä. Microsoft kehitti omaan kilpailevaan selaimeensa, Internet Explorerin versioon 3, oman versionsa JavaScriptistä, nimeltään JScript. Kilpailu yritysten kesken samankaltaisten kielten kehittämisessä edesauttoi yhteistyötä Netscapen ja teknologiayritys Sun Microsystemsin välillä. Yhtiöt laittoivat yhdessä aluille European Computer Manufacturers Associationin (ECMA) ja aloittivat JavaScriptin standardoinnin. [6] JavaScriptin rakenteellisia ongelmia ei täysin ymmärretty aluksi, vaan kokemattomat ja ymmärtämättömät kehittäjät käyttivät nykyään huonoja käytänteitä, kuten globaaleja muuttujia, huoletta.

JavaScript on C-, C++- ja Java-kielten tapaan imperatiivinen ohjelmointikieli, eli siinä käytetään esimerkiksi kyseisissä kielissä usein esiintyviä *if*- ja *while*-lauseita, mutta JavaScript eroaa näistä kielistä merkittävästi muutamalla tavalla. Objektit ovat JavaScriptissä lähempänä avain-arvo-pareja sisältäviä assosiatiivisia taulukoita kuin C++:n tai Javan objekteja, vaikka ne kulkevatkin samalla nimellä. [8] Toisin kuin edellä mainitut käännetyt ohjelmointikielet, JavaScriptiä ei käännetä ollenkaan, vaan verkkoselain lukee sille annetun skriptin ja ajaa sen. Toteutustavassa on sekä heikkoutensa että vahvuutensa: JavaScript on kielenä helposti lähestyttävä sen yksinkertaisuuden ja helppokäyttöisyytensä takia, mutta toisaalta juuri näiden ominaisuuksien takia se on myös ansainnut maineensa haavoittuvana kielenä.

Haavoittuvuuksien, esimerkiksi myöhemmin esiteltävän *Cross-Site Scripting (XSS)*-hyökkäysten minimoimiseksi JavaScript hyödyntää *Same-Origin Policy (SOP)* -pääsynvalvontaa: vain samasta alkuperästä tulevaa JavaScript koodia voidaan ajaa samalla sivustolla, ellei toisin ole määritelty. Alkuperä on tässä tapauksessa verkkoportti, URL tai domain. [7] Tapoja käyttää kolmannen osapuolen JavaScript-paketteja on kehitetty varmistamaan, että vain turvallinen ja sivuston laatijan hyväksymät skriptit ajetaan sivustolla. Tämä varmistetaan lataamalla paketti suoraan tai merkitsemällä paketti ladattavaksi sivuston lataushetkellä.

Document object model (DOM) eli dokumentti-objekti-malli on yleisesti HTML- ja XML-kielillä käytetty ohjelmointirajapinta. *Dokumentilla* dokumentti-objekti-mallin nimessä ei tarkoiteta aivan nimensä mukaisesti verkkosivun dokumentteja, vaan niistä saatavaa da-

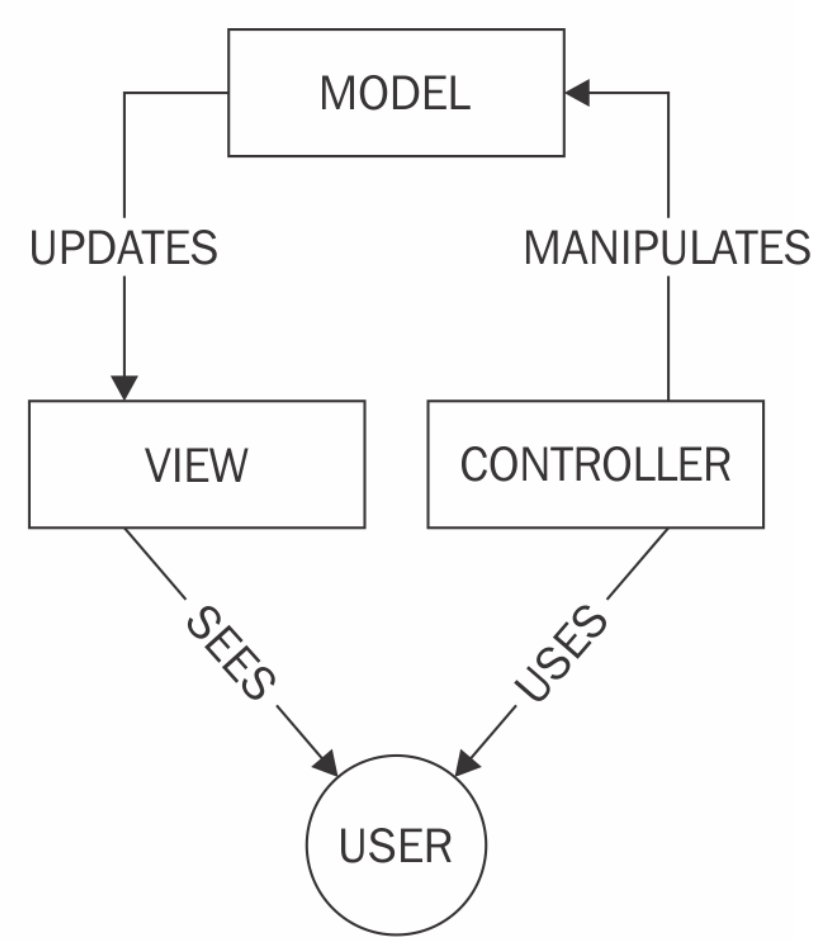
taa. Verkkosivua ladatessa DOM muuntaa sivun mukana ladattavista dokumenteista objekteja, joita voidaan käsitellä Javascriptilla. [6] *Objekti* tai muista ohjelmointikielistä tunnettu termi olio on kuin suljettu kasa dataa, jota voidaan käsitellä objektin määritellyn rajapinnan kautta. Javascriptissä esiintyy kolmenlaisia objekteja:

- ohjelmoijan itse kehittämiä objekteja
- JavaScriptiin sisäänrakennettuja objekteja, kuten *Array*
- selaimen tarjoamia objekteja. [6]

Sivua ja sen sisältämiä objekteja voidaan kutsua esimerkiksi DOM:n luoman *document*-objektin kautta.

3.1 MVC-MALLIPOHJA

Suurin osa nykypäivän suosituista JavaScript-työkaluista, AngularJS mukaan lukien, käyttävät *Model-View-Controller (MVC)* -mallipohjaa pohjaratkaisuna [13]. MVC-mallipohja soveltuu hyvin verkkokehitykseen, sillä sen sisältämät komponentit yhdistävät monet verkkokehityksen eri kohdissa tarvittavat elementit. MVC-mallipohjan toimintaa on havainnollistettu kuvassa 3.1.



Kuva 3.1: MVC-mallipohja [11]

MVC-mallipohja toimii yleensä syklissä: käyttäjä lähettää tietoa, jonka ohjain (*controller*) ottaa vastaan ja toimii sille asetetun toimintamallin mukaan. Ohjaaja lähettää tarvittavat tiedot mallille (*model*), joka sijaitsee palvelimella tai sillä on yhteys palvelimelle. Malli päivittää käyttäjän päätelaitteen näkymän (*View*) ohjaajan antamien tietojen mukaan.

Mallin päätehtävänä MVC-mallipohjassa on piirtää käyttäjän päätelaitteelle tarvittava tieto ja hallinnoida tarvittavia visuaalisia elementtejä, kuten myös hoitaa pintatason loogisia operaatioita. Malli rakennetaan usein tiedon abstrahoinnin näkökulmasta: sen on tarkoitus näyttää käyttäjälle vain tarpeellinen tieto. Mallin päätehtävät ovat siis pääsynvalvonta ja tiedon vahvistaminen [15]. Mallit rakennetaan usein käyttämällä relatiivisia tauluja tai XML-kieltä. [11]

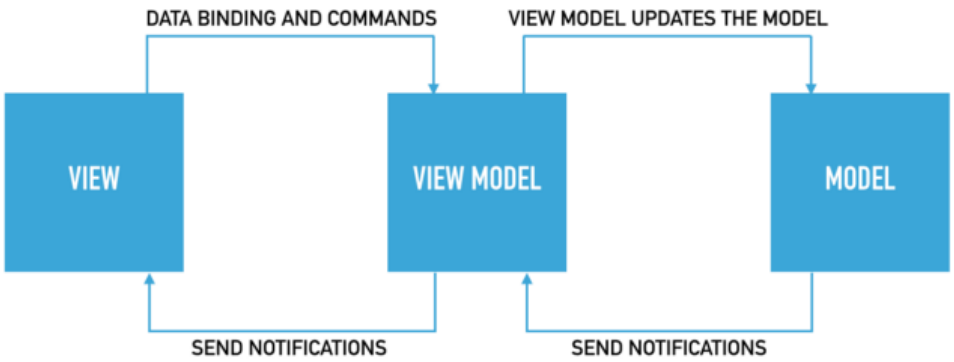
Näkymä käsittelee käyttäjän päätelaitteella näkyvää graafista käyttöliittymää, eli kaikkia sen komponentteja: nappeja ja muita graafisia elementtejä. Kun visuaalinen näkymä erotetaan varsinaisesta toimintalogiikasta, voidaan sitä myös muokata ilman mahdollisia toimintalogiikkaan vaikuttavia muutoksia [15].

Ohjain toimii MVC-mallipohjan kolmesta komponentista tapahtumien valvojana. Tapahtumat voivat syntyä joko käyttäjän toimesta tai järjestelmän aiheuttamina. Kuten kuvassa 1 on esitetty, ohjain saa päätelaitteelta tapahtuman vastaan, jonka se ohjaa mallille toimintaohjeiden kera.

On suositeltu käytettävän lihavan ohjaimen periaatetta suunniteltaessa MVC-mallipohjan mukaista ohjelmistoa. Lihavalla ohjaimella tarkoitetaan sitä, että kaikki tiedonhallinta tapahtuu ohjaintasolla. [15] Menettelytapa mahdollistaa paremman tietoturvan ohjelmalle, sillä tietoa käsitellään mahdollisimman pienellä määrällä komponentteja. Tässä yhteydessä puhutaan myös ohuesta ja lihavasta päätelaitteen ohjelmasta.

3.2 MVVM-MALLIPOHJA

Vaikka suurin osa JavaScript-työkaluista käyttää MVC-mallipohjaa, usein käytetään myös *Model-View-ViewModel*-mallipohjaa. Tästä käytetään yleensä lyhennettä *MVVM*. MVVM-mallipohjaa päädyttiin käyttämään JavaScriptin kehityksessä sen näkymämallin (*ViewModel*) luoman väliseinän takia: näkymämalli toimii välittäjänä palvelimella toimivan mallin (*Model*) ja käyttäjän omalla päätelaitteella toimivan näkymän (*View*) välillä. Täten käyttäjän päätelaite käsittelee vain sen tarvitsemaa dataa. [13] MVVM-mallin toimintaa on havainnollistettu kuvassa 3.2.



Kuva 3.2: MVVM-mallipohja [12]

MVVM-mallipohjan malli toimii lähes samalla tavalla kuin MVC-mallipohjan, mutta sen näkymä ja näkymämalli toimivat huomattavan eri lailla: kaksisuuntaista sitomista voidaan hyödyntää tässäkin tapauksessa, nyt mallin ja näkymän välillä.

MVVM-mallipohjan näkymä hallitsee ohjelman visuaalisia elementtejä, kuten nappeja ja valikoita sekä monimutkaisempia elementtejä. Toisin kuin MVC-mallipohjassa, näkymä hoitaa myös ohjelman toimintalogiikkaa, jota ennen hoiti ohjelman ohjaaja. [13]

Näkymämalli toimii väliseinänä mallin ja näkymän välillä tarjoten mallille tavan hyödyntää kaksisuuntaista sitomista. Näkymämalli on päävastuussa käyttäjän päätelaitteen ja palvelimen välisestä tietoliikenteestä pitäen huolen siitä, ettei käyttäjä saa palvelimelta enempää tietoa kuin käyttäjä tarvitsee sekä pitäen myös huolen siitä, ettei palvelimen tarvitse tehdä turhaa työtä käyttäjän haluaman tiedon lähettämiseen. Näkymämalli toimii siis välittäjänä näkymän ja mallin välillä, toisin kuin MVC-mallipohjassa, jossa kaikki kolme mallipohjan osaa toimivat rinnakkain. Lisäksi näkymämalli sisältää tarvittavat toiminnallisuudet käyttäjän näkymää varten. Näkymämalli on toteutettu AngularJS-työkalussa käyttäen näkyvyysalueita (*scope*). [13] Näkyvyysalueet on käyty tarkemmin läpi luvussa 5.

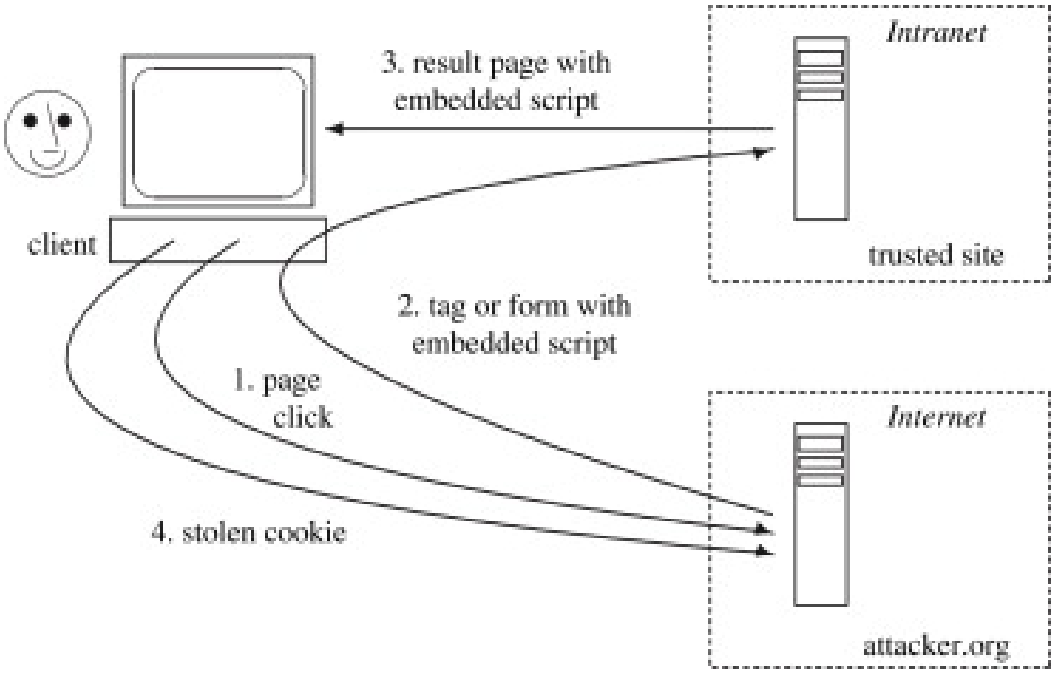
4. JAVASCRIPT TIETOTURVANÄKÖKULMASTA

Tietoturvallisuus on erittäin tärkeä näkökulma ohjelmointikielen käytössä, sillä 88,45% maailman top 10000 listan verkkosivuista käyttivät ainakin yhtä JavaScript-liitännäistä. Sivustojen määrä on tuplaantunut vuodesta 2010. [7] JavaScriptin tietoturva on erittäin laaja alue, joten tässä kandidaatintyössä on tutkittu yleisimpiä ja samalla vaarallisimpia tietoturvavirheitä ja -haavoittuvuuksia.

JavaScriptin tietoturvassa on tehty suuria harppauksia sen kehitysvuosien aikana. Verkkoteknologioiden kehitys on aiheuttanut uusia ongelmia ja myös auttanut ratkaisemaan vanhempia JavaScriptin ongelmia. JavaScriptin keskeisiä ongelmia tietoturvan näkökulmasta ovat sen yksinkertaisuus ja mahdollisuus suorittaa skriptejä kääntämättä niitä. [6] Nämä ovat molemmat myös suuria etuja verrattuna muihin raskaampiin ja vanhempiin ohjelmointikieliin, kuten C++:an verrattuna.

4.1 CROSS-SITE SCRIPTING

Cross-Site Scripting (XSS) on hyökkäystapa, jossa hyökkääjä hyödyntää uhrin järjestelmän haavoittuvuutta ajamalla omaa koodiaan uhrin järjestelmällä. XSS-hyökkäyksiä on monia eri tyyppisiä. Tyypillisimmät hyökkäystavat ovat heijastettu XSS-hyökkäys, josta esimerkki kuvassa 4.1, talletettu XSS-hyökkäys sekä DOM-pohjainen hyökkäys. Yksinkertaisessa XSS-hyökkäyksessä verkkopalvelin antaa hyökkääjän tallentaa haitallista koodia palvelimelle, minkä jälkeen uhrin selain on alttiina suorittamaan hyökkääjän koodin, mikäli uhri navigoi saastuneelle sivustolle. [7] XSS-hyökkäyksiä käytetään muun muassa kaappaamaan käyttäjän istuntoavain, jolloin hyökkääjä saa pääsyn käyttäjän istuntoon.



Kuva 4.1: heijastettu XSS-hyökkäys [10]

Funktion *eval()* toteutuksen haavoittuvuuksien hyödyntäminen on toinen kriittinen XSS-hyökkäys. Funktio ottaa parametrinaan vastaan *string*-tyyppisen muuttujan, joka voi sisältää käytännössä mitä vain. Funktio odottaa aloitusparametrin sisältävän lausekkeen, käskyn tai joukon käskyjä. Funktiolla voi esimerkiksi laskea yhteenlaskuja:

```
eval('2 + 2');
```

Funktiolla voi kuitenkin tehdä paljon muutakin, kuten ajaa haitallista koodia, jonka hyökkääjä on saanut injektoitua sivustolle. Mozillan dokumentaatio *eval()*-funktiosta kehottakin olemaan käyttämättä sitä. Haavoittuvuuksien lisäksi dokumentaatiossa lisätään, että *eval()* on hitaampi kuin vaihtoehtoiset funktiot, esimerkiksi *Function()*. [4] Funktio on kuitenkin erittäin laajassa käytössä sen lukuisista haittapuolista huolimatta.

Heijastetussa XSS-hyökkäyksessä hyökkääjä lähettää palvelimelle haitallista koodia jollakin tavalla, kuten lataamalla palvelimelle koodia sisältävän tiedoston tai antaa uhrille linkin, joka sisältää piilotettua JavaScript-koodia [16]. Saatuaan uhrin avaamaan linkin, ja täten suorittamaan haitallisen koodin, on hyökkäys suoritettu. Toisin kuin talletetussa XSS-hyökkäyksessä hyökkäys käynnistyy heti uhrin avattua linkki tai palvelimen saatua tiedosto.

Kolmantena XSS-hyökkäystapana toimii *DOM-pohjainen XSS-hyökkäys.* DOM-pohjaisessa hyökkäyksessä haitallinen koodi on valmiiksi sijoitettuna palvelimelle joko ohjelmointivirheen tai aiemman hyökkäyksen ansiosta. DOM-pohjainen hyökkäys toimii kuten

heijastettu hyökkäys, mutta DOM-pohjaisessa hyökkäyksessä hyökkääjän ei tarvitse syöttää haitallista koodia palvelimelle [16].

OWASP listasi XSS-haavoittuvuudet vuonna 2017 seitsemänneksi kriittisimmäksi tietoturvaongelmaksi verkkokehityksessä yleisesti. [3] Vuoden 2013 tutkimuksessa OWASP sijoitti XSS-ongelmat sijalle 3 listallaan. Vaikka XSS sijaitsee yleisellä listalla vasta sijalla seitsemän, ovat XSS-haavoittuvuudet kaikkein laaja-alaisin mahdollinen hyökkäysvektori. XSS-haavoittuvuuksien määrä kasvaa jatkuvasti verkkosivujen sosiaalisten komponenttien määrän kasvaessa: käyttäjät saavat ladata sivustoille enemmän ja enemmän mahdollisesti haitallista materiaalia luoden lisää mahdollisuuksia hyökkääjille. [6].

XSS-hyökkäykseltä suojautumisessa helpoimmaksi keinoksi on havaittu käyttäjien syötteiden tarkistus, eli niiden sisältämän tiedon muuntaminen selkotekstiksi. Samalla estetään mahdollisen haitallisen koodin suorittaminen vahingossa. Tämä lähestymistapa ei kuitenkaan riitä suojaamaan sivustoa kaikilta HTML-pohjaisilta hyökkäyksiltä, sillä on vaikeaa erotella haitallinen HTML-koodi viattomasta HTML-koodista. [6] Muita tapoja suojautua XSS-hyökkäyksiltä ovat *Content Security Policy (CSP)* sekä *Same-Origin Policy (SOP)*, jolla estetään ulkopuolisten skriptien ajaminen sivustolla. CSP on käytännössä SOP:n päivitetty versio, jota käytetään silloin, kun selain tukee sitä ja se on otettu käyttöön palvelimen HTTP-tunnisteessa. Mikäli selain ei kykene hyödyntämään CSP:tä, se sivuutetaan ja selain käyttää SOP:tä. [14]

4.2 EPÄTURVALLINEN SARJOITTAMISEN PURKU

Epäturvallinen sarjoittamisen purku (insecure deserialization) aiheutuu JavaScriptissä pääasiassa JSON-tyyppisten tekstipakkaustiedostojen sarjoittamisesta epäturvallisia funktioita, kuten aikaisemmin mainittua *eval()*-funktiota, käyttäen. Epäturvallisessa sarjoittamisessa tietoa, jonka alkuperä ei ole tiedossa tai jonka alkuperä on epäturvallinen, käsitellään kuin luotettavaa tietoa.

Node.js-paketin moduulissa *node-serialize* sisäänrakennettuna funktiona oleva *unserialize()* on yksi esimerkki epäturvallisesta sarjoittamisesta. Funktio saa verkkokutsusta parametrikseen evästeen, jonka sisältämälle tiedolle kutsutaan myöhemmin jo aiemmin tutuksi tullutta *eval()*-funktiota. Käyttämällä saman paketin *serialize()*-funktiota, on mahdollista sarjoittaa JavaScript-objekti, jonka *unserialize()* hyväksyy parametrikseen. Kun sarjoitettu objekti annetaan parametrina funktiolle, sen sisältämä haitallinen koodi suoritetaan sarjoituksen purun yhteydessä. [5]

4.3 CROSS SITE REQUEST FORGERY

Cross Site Request Forgery (CSRF tai XSRF) eli sivustolle tulevan pyynnön väärentäminen mielletään joidenkin kehittäjien mielessä samaksi kuin XSS-hyökkäys. Kokematon verkkokehittäjä voi myös mieltää sivustonsa olevan suojassa CSRF-hyökkäykseltä samoilla mekanismeilla kuin XSS-hyökkäykseltä. Tämä ei kuitenkaan ole käytännössä totta, sillä vaikka CSRF-hyökkäykset ovat samantyyppisiä kuin XSS-hyökkäykset vaaditaan niiltä suojautumiseen erityisiä suojauskeinoja [17].

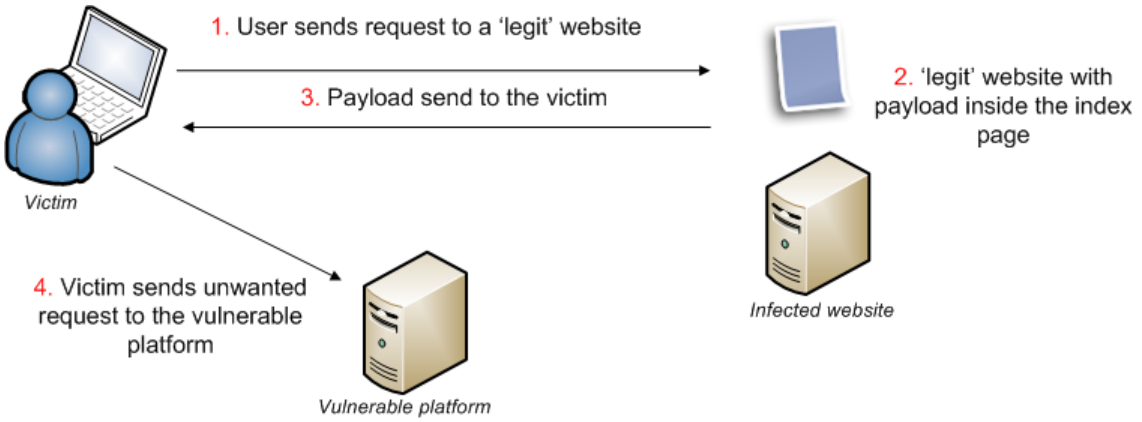
OWASP listasi vuonna 2013 CSRF-hyökkäyksen sijalle kahdeksan kymmenen kriittisimmän tietoturvaongelman listauksessaan, mutta sitä ei ole sisällytetty vuoden 2017 listaukseen. Syyksi tähän annettiin monien työkalujen sisäänrakennetut CSRF-hyökkäyksien vastatoimet. Näistä vastatoimista huolimatta CSRF-haavoittuvuuksia löytyi 5% testatuista ohjelmistoista. [3]

CSRF-hyökkäyksiä on monenlaisia; niillä voidaan tehdä pientä kiusantekoa, kuten viestin lähetys keskustelupalstalle. Myös suurempia haavoittuvuuksia hyödyntäviä tekoja voidaan CSRF-hyökkäyksillä mahdollistaa, kuten käyttäjän tietojen vaihto, esimerkiksi salasanan ja käyttäjätunnuksen vaihto. CSRF-hyökkäyksen onnistumiseksi vaaditaan yleensä kaksi tekijää: ensiksi käyttäjän on täytynyt unohtaa kirjautua tai jättänyt kirjautumatta ulos sivustolta, jolle hyökkääjä on suunnitellut hyökkäyksensä, toiseksi käyttäjän täytyy navigoida hyökkääjän muokkaamalle sivustolle [18].

CSRF-hyökkäykset toimivat useimmiten samalla periaatteella: hyökkääjä huijaa käyttäjää navigoimaan sivustolle, johon on upotettu haitallinen URL-osoite esimerkiksi <img>-elementin sisään. CSRF-hyökkäystä on havainnollistettu kuvassa 4.3. Kun sivusto latautuu käyttäjän päätelaitteella, sivusto yrittää myös ladata kuvan osoittesta <img>-elementin sisällä GET-pyynnöllä. [17] Mikäli elementin sisällä oleva osoite sisältää jotakin muuta, kuten esimerkiksi komentoja:

```
<img src=www.example.com/changepassword.php?newpass=salasana>
```

Yllä olevassa yksinkertaisessa GET-pyynnöllä toteutetussa CSRF-hyökkäyksessä hyökkääjä yrittää vaihtaa käyttäjän salasanan asettamalla changepassword.php-sivun newpass-parametrin arvoksi salasana. Mikäli sivustolla ei ole sisäänrakennettua CSRF-hyökkäyssuojaa, saa hyökkääjä vaihdettua käyttäjän salasanan haluamakseen ja täten pääsyn käyttäjän profiiliin.



Kuva 4.3: CSRF-hyökkäys [19]

CSRF-hyökkäyksiä voidaan myös toteuttaa JavaScriptin avulla.

5. ANGULARJS

5.1 ANGULARJS YLEISESTI

AngularJS-työkalu on tuotekehitystä suoraviivaistava verkkosovellusten kehitystyökalu [24]. AngularJS-työkalun päätavoitteina kehityksessä olivat vähentää kirjoitettavan koodin määrää ja tehdä ohjelmiston kehittämisestä sulavampaa erinäisten valmiiksi tehtyjen ominaisuuksien myötä, kuten suodattimien ja valmiiksi tehtyjen moduulien.

JavaScriptissä kehittäjällä on neljä tapaa ratkaista ohjelman komponentin sisäiset riippuvaisuudet: [13] *new*-operaattori, joka luo objektin rakentajaa käyttäen uuden olion. Tällä operaattorilla luotua objektia on kuitenkin vaikeaa muuttaa ohjelman ollessa käynnissä. *Globaalit muuttujat*, joita ei suositella käytettävän, tekevät ohjelman koodista vaikealukuisempaa ja ovat alttiita huolimattomuusvirheille esimerkiksi tahattoman ylikirjoituksen vuoksi. Lisäksi globaaleja muuttujia käyttävää moduulia on erittäin vaikeaa suunnitella toimivaksi moneen eri käyttötarkoitukseen. *Joustavuus ja vaihtokelpoisuus* ovat sinänsä hyviä tapoja ratkaista komponentin riippumattomuus -ongelma, mutta ohjelman pitää toimiakseen oikein pystyä pääsemään käsiksi moduulin käyttämään palveluun ja sen luomiin objekteihin. [13]

AngularJS-työkalu on suunniteltu neljännen tavan, eli *testauksen* näkökulmasta, sillä ohjelman kehittäjät kokivat sen olevan näistä neljästä ylivoimaisesti hyödyllisin. Testauksen näkökulmasta kehitetyllä työkalulla on monia etuja verrattuna edellisiin kolmeen tapaan lähestyä moduulin riippumattomuutta, kuten virheiden helpompi ja nopeampi havaitseminen sekä tarvittavan koodin määrän vähentyminen, sillä kehittäjän tarvitsee vain ratkaista aikaisemmin luodun testin havaitsema ongelma. AngularJS:llä voidaan testata kaikkia ohjelman osia yksitellen eristyksissä toisistaan yksikkötestausvaiheessa, kuten myös myöhemmin ohjelman integraatiovaiheessa, kun ohjelma liitetään haluttuun toimintaympäristöön integraatiotestien muodossa. [13]

AngularJS-työkalulla luotu ohjelma koostuu useimmiten yhdestä tai useammasta moduulista, ohjaimesta (*controller*), mallista, reiteistä, näkymistä, pohjista (*template*) ja käskyistä (*expression*), suodattimista (*filter*), direktiiveistä, näkyvyysalueista (*scope*) ja tiivistelmästä. [13] Yksittäiset AngularJS-työkalulla luodut sovellukset eivät voi suoraan keskustella keskenään tai vaikuttaa toistensa toimintaan [13], sillä tämä avaisi mahdollisen tietoturva-aukon hyökkääjien käyttöön.

AngularJS-työkalussa on sisäänrakennettuna paketti täynnä monenlaisia tarpeellisia suodattimia, kuten päivämäärän, pienten kirjainten ja isojen kirjainten suodatintoimintoja. Kustomoitujen suodattimien luonti on myös mahdollista. [13] Kustoimoitujen suodattimien luonti mahdollistaa niiden uudelleenkäytön muualla sovelluksessa samaan tapaan kuin funktioita voidaan käyttää uudelleen monessa eri osassa sovellusta. Suodattimet toimivatkin hieman samaan tapaan kuin funktiot.

Näkyvyysalue sisältää kaikki tietyssä kontekstissa määritellyt muuttujat ja funktiot [13], samaan tapaan kuin esimerkiksi tavanomaisessa JavaScript-tiedostossa. Näkyvyysalueet voidaan periyttää suoraan prototyyppien kautta. Tämä periytys toimii samalla tavalla kuin luokkien periytys yleensä ohjelmistomaailmassa; abstrahoitu pohja peritään, jonka pohjasta luodaan uusi luokka, tai tässä tapauksessa näkyvyysalue. Tämä menettelytapa vähentää merkittävästi kirjoitetun koodin määrää, vähentää toistoa sekä turhaa koodin uudelleenkirjoitusta. [13]

Näkyvyysalueita hyödynnetään AngularJS-työkalussa myös mallin ja näkymän tahdistamiseen suunnitellulla mekanismilla nimeltään *dirty checking*. Sillä etsitään kaikki oleelliset näkyvyysalueet näkymälle ja tarkistetaan, onko mikään näkyvyysalueiden sisäinen muuttuja muuttunut. Mikäli näkymän ja näkymämallin välillä havaitaan ero, tahdistuslogiikkaa päivittää näkymän viimeisimpään versioon. [13] Tämä toimii molempiin suuntiin kaksisuuntaisen sidonnan avulla.

Näkymämalli onkin toteutettu kaksisuuntaisen sidonnan ohella näkyvyysalueiden kautta, mikä rajoittaa kaksisuuntaisen sidonnan näkyvyysalueen funktioista ja muuttujista vain näkyvyysalueessa määritellyille [13], ei siis sisäänrakennetuille ilman erillistä periyttämistä. Kaksisuuntainen sidonta mahdollistaa myös sovelluksen luonnin kirjoittamatta riviäkään JavaScriptiä käyttäen sisäänrakennettuja funktioita.

5.2 ANGULARJS TIETOTURVANÄKÖKULMASTA

Paras ja yksinkertaisin tapa kehittää AngularJS-työkaluun luotujen sovellusten tietoturvaa on seurata kehittäjien ilmoituskanavia ja varautua päivittämään työkalu uusimpaan versioon uusien haavoittuvuuksien ilmetessä. Päivityksetkään eivät kuitenkaan takaa työkalulla luodun sovelluksen turvallisuutta täysin, vaan kehittäjän täytyy tutustua työkalun dokumentaatioon ja sen sisäänrakennettuihin turvallisuusominaisuuksiin. Kaikissa työkaluissa sisäänrakennettuja turvallisuusominaisuuksia ei edes ole, vaan näissä tapauksissa kehittäjän on tarpeellista huolehtia turvallisuudesta itse. AngularJS-työkaluun on sisällytetty monia eri turvallisuusmekanismeja suojaamaan luvussa 4 esitellyiltä tietoturvaongelmilta, muttei kuitenkaan kaikilta.

AngularJS-työkalu estää kehittäjää suoraan kutsumasta eval()-funktiota, poistaen erään aiemmin luvussa 4 käsitellyn JavaScriptin sisäänrakennetun tietoturva-aukon. AngularJS käyttää kuitenkin sisäisesti eval()-funktiota eräiden sisäänrakennettujen toimintojen kanssa [25]. AngularJS käyttää myös turvallisempaa, noin 30% hitaampaa versiota eval()-funktiosta. Tämä on tyypillistä tietoturvamaailmassa: parannettu turvallisuus johtaa usein hitaampaan toimintaan ja toisinpäin. Tämä johtaa valitettavan usein kehittäjien tekemiin kompromisseihin, joissa turvallisuudesta leikataan nopeamman ohjelmiston toivossa.

AngularJS-työkalun aiemmat versiot sisälsivät lausekehiekkalaatikkotyökalun, jonka avulla kehittäjä pystyi luomaan lausekkeita dynaamisesti. Työkalulla oli mahdollista lisätä koodia pohjaan tai lausekkeeseen käyttäjän syötteestä, avaten oven mahdollisille XSS-hyökkäyksille puhdistamattoman syötteen kautta. Tämä työkalu on poistettu versiosta 1.6 eteenpäin, sillä jotkin kehittäjät luulivat, että työkalulla luodut lausekkeet olivat suojattuja XSS-hyökkäyksiltä [23]. AngularJS-työkalun kehittäjät kokivat työkalulla luotujen sovellusten parhaaksi poistaa hiekkalaatikkotyökalu kokonaan, sillä siitä koitui kokemattomille kehittäjille turhia sudenkuoppia. Hiekkalaatikkotyökalun poistamisen sivuvaikutuksena AngularJS-työkalun toiminta nopeutui. [23]

AngularJS-työkalu erottaa HTML-, CSS- ja JavaScript-tiedostot toisistaan, mikä johtaa rivinsisäisten skriptien estoon, parantaen ohjelmiston ajoaikaista turvallisuutta [25]. Rivinsisäisen skriptit

AngularJS-työkalu sisältää myös sisäänrakennetun XSS-hyökkäyssuojan. XSS-hyökkäykset on käyty tarkemmin läpi luvussa 4. Tämä suoja ei ole kuitenkaan täydellinen:

mikäli hyökkääjällä on pääsy AngularJS-työkalulla luotuihin ohjelmiston pohjiin tai käskyihin, on XSS-hyökkäys mahdollinen työkalun versiosta riippumatta [25]. Tämä johtuu AngularJS-työkalun toteutusteknisistä syistä.

XSS-hyökkäykset ovat mahdollisia myös, mikäli ohjelmisto merkitsee huolimattoman ohjelmoinnin myötä epäluotettavaa dataa luotetuksi funktiokutsulla, esimerkiksi *$sce.trustAsHtml*-kutsun kautta [25]. Tällöin hyökkääjä voi luvun 4 esimerkin tapaan syöttää omaa haitallista koodiaan ohjelmistolle, jonka se suorittaa.

AngularJS sisältää myös luvussa 4 esitellyn JSON-tiedostojen purun kautta tapahtuvien hyökkäysten varalta suojauksen. Myös CSRF-hyökkäyksiä varten on mahdollista suojautua AngularJS-työkalulla luoduissa sovelluksissa sisäänrakennetuilla suojausmekanismeilla. [25]

Lokaalien tietovarastojen haavoittuvuus ei ole varsinaisesti AngularJS-työkalun ominainen ongelma, vaan yleisesti lokaalien tietovarastojen turvallisuuteen liittyvä haitta, mutta kehittäjä voi avata sovelluksen tietovaraston avoimeksi hyökkääjälle, mikäli kehittäjä valitsee tallennustavaksi lokaalin tallennuksen esimerkiksi pilven kautta. Tällöin muiden käyttäjien tiedot saattavat päätyä hyökkääjän käytettäviksi ilman, että hyökkääjän tarvitsee autentikoida itsensä. [25]

6. YHTEENVETO

Nykypäivän ohjelmistokehittäjä ei voi luottaa pelkkiin työkaluihin ja omiin tietotaitoihinsa tietoturvallisuuden alati kiihtyvässä kilpajuoksussa. Vaikka työkalut kehittyvät jatkuvasti uusien turvallisuusominaisuuksien ja haavoittuvuuksien paikkaamisen muodossa, eivät nekään aina riitä pitämään ohjelmistoa turvallisena. Varsinkin tulevaisuudessa jatkuva oppiminen ja uusien uhkien kartoittaminen ja tunnistaminen tulevat olemaan tärkeitä seikkoja verkkokehityksen tietoturvallisuuden maailmassa, sillä uudet teknologiat tuovat mukanaan uusia haasteita turvallisuuden saralla.

JavaScript-kieli sisältää monia sisäänrakennettuja tietoturva-aukkoja, joihin kokematon kehittäjä helposti putoaa. Verkkokehityksen kasvavan monimutkaisuuden vuoksi jokaisen kehittäjän tulisi ainakin tiedostaa mahdolliset yleisimmät ongelmat valitsemansa kehitystyökalun tai ohjelmointikielen osalta. Tiedostamalla ongelmien olemassaolo voidaan myös ehkäistä virheiden syntyä ohjelmoinnin tasolla.

AngularJS-työkalussa on otettu huomioon suurimpia tietoturvallisuusongelmia nykypäivän verkkokehityksessä, kuten XSS-hyökkäykset, ja niitä vastaan on kehitetty suojamekanismeja. Nämä suojamekanismit eivät kuitenkaan täysin riitä korvaamaan ohjelmistokehittäjän tietotaitoa tietoturvallisuudesta, vaan kehittäjän täytyy myös itse pystyä tunnistamaan mahdolliset tietoturvaongelmat ohjelmistossa. Työkalun tietoturvaominaisuuksia tulee hyödyntää, mutta niihin ei pidä luottaa täysin. Hyvänä esimerkkinä tästä toimii AngularJS-työkalun sisäänrakennettu XSS-suoja, joka on teoriassa luotettava, mutta sekin pystytään kiertämään.

Kuten minkä tahansa laitteen tai teknologian kanssa, myös JavaScriptin ja verkkoteknologioiden kanssa oikein käytetty työkalu toimii ohjelmistokehittäjän arvokkaana apuna. Puutteellisen tietotaidon omaavaa tai huolimatonta kehittäjää eivät pelkät työkalut pelasta tietoturvauhkien armottomassa maailmassa.

LÄHTEET

[1] Hedin, D., Bello, L., Sabelfeld, A., Mälardalens högskola, Akademin för innovation, design och teknik & Inbyggda system 2016, "Information-flow security for JavaScript and its APIs", *Journal of Computer Security,* vol. 24, no. 2, pp. 181-234.

[2] Phung, P., Sands, D. & Chudnov, A. 2009, "Lightweight self-protecting JavaScript", ACM, *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pp. 47-60.

[3] OWASP Foundation, (2017). *OWASP Top 10 – 2017: The Ten Most Critical Web Application Security Risks.* 1st ed. [pdf] Saatavilla: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf [luettu 18.2.2019]

[4] Mozilla and individual contributors, (2018). *MDN web docs*. [online] Saatavilla: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval [luettu 15.2.2019]

[5] Ajin Abraham. Saatavilla: https://www.exploit-db.com/docs/english/41289-exploiting-node.js-deserialization-bug-for-remote-code-execution.pdf [luettu 9.2.2019]

[6] Keith, J., Sambells, J. & Books24x7, I. 2010, DOM Scripting: Web Design with JavaScript and the Document Object Model, Second Edition, 1;2nd; edn, friends of ED, New York.

[7] Bielova, N. 2013, "Survey on JavaScript security policies and their enforcement mechanisms in a web browser", *Journal of Logic and Algebraic Programming,* vol. 82, no. 8, pp. 243-262.

[8] Flanagan, D. 2006, *JavaScript: The Definitive Guide*, O'Reilly Media, Incorporated.

[9] Tarasiewicz, P. & Böhm, R. 2014*, AngularJS*, Brainy Software, Vancouver.

[10] Gollmann, D. 2008, "Securing Web applications", *Information Security Technical Report,* vol. 13, no. 1, pp. 1-9.

[11] https://www.oreilly.com/library/view/dependency-injection-in/9781787121300/8a977b34-5804-4edf-8ec1-4f07e682f5f2.xhtml

[12] https://medium.com/@husayn.hakeem/android-by-example-mvvm-data-binding-introduction-part-1-6a7a5f388bf7

[13] Tarasiewicz, P. & Böhm, R. 2014, *AngularJS,* Brainy Software, Heidelberg, Germany.

[14] Mozilla and individual contributors, (2018). *MDN web docs*. [online] Saatavilla: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP [luettu 15.2.2019]

[15] Pop, D. & Altar, A. 2014, "Designing an MVC Model for Rapid Web Application Development", *Procedia Engineering,* vol. 69, s. 1172-1179.

[16] Kour, H. & Sharma, L.S. 2016, "Tracing out Cross Site Scripting Vulnerabilities in Modern Scripts", *International Journal of Advanced Networking and Applications,* vol. 7, no. 5, s. 2862.

[17] Kombade, R.D. & Meshram, B.B. 2012, "CSRF Vulnerabilities and Defensive Techniques", *International Journal of Computer Network and Information Security,* vol. 4, no. 1, s. 31-37.

[18] Shema, M. & Books24x7, I. 2012, Hacking Web Apps: Detecting and Preventing Web Application Security Problems, Elsevier Science, Rockland, MA.

[19] Paul Amar, (2014). Saatavilla: http://paulsec.github.io/bsides-london-2014/#/4 [luettu 7.4.2019]

[20] Le Roux, Y. 1993, "Information security - The CIA model", Director, [Online], , pp. 53.

[21] Lundgren, B., Möller, N., Skolan för arkitektur och samhällsbyggnad (ABE), Filosofi, KTH & Filosofi och teknikhistoria 2019;2017;, "Defining Information Security", Science and Engineering Ethics, vol. 25, no. 2, pp. 419-441.

[22] Laihonen, H., Hannula, M., Helander, N., Ilvonen, I., Jussila, J., Kukko, M., Kärkkäinen, H., Lönnqvist, A., Myllärniemi, J., Pekkola, S., Virtanen, P., Vuori, V. & Yliniemi, T. 2013, Tietojohtaminen, Tampereen teknillinen yliopisto, Tietojohtamisen tutkimuskeskus Novi.

[23] AngularJS. Saatavilla: https://blog.angularjs.org/2016/09/angular-16-expression-sandbox-removal.html [luettu 12.4.2019]

[24] AngularJS dokumentaatio. Saatavilla: https://docs.angularjs.org/guide/introduction [luettu 28.3.2019]

[25] AngularJS dokumentaatio. Saatavilla: **https://docs.angularjs.org/guide/security** [luettu 28.3.2019]